Arabian Journal for Science and Engineering (2023) 48:10441–10455
https://doi.org/10.1007/s13369-023-07655-6

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Hardware Design of FPGA-Based Embedded Heuristic Optimization Technique for Solving a Robotic Problem: IC-PSO

Serkan Dereli[1] · **Raşit Köker**[2]

Received: 14 August 2022 / Accepted: 25 January 2023 / Published online: 11 February 2023

Abstract
In this study, a hardware design that runs the PSO algorithm at the micro level is carried out using FPGA technology, which facilitates prototyping and testing of integrated circuit systems. In this way, a high rate of acceleration has been achieved for swarm algorithms that perform stochastic search and whose calculation time is not suitable for real-time systems. For this, the inverse kinematics problem, which is used in the robotics field and which forms the basis of the robot control system, is solved for a 7-joint serial robot manipulator. Since the study aims to compare software-based calculations and hardware-based calculations, the results are presented in a comparative way with the results obtained with Matlab software. The tests have been performed in the study revealed two important situations. Firstly, the biggest handicap of algorithms such as PSO that reach a result by searching in a certain solution space is that the solution times are not suitable for real-time applications. The other is that FPGA can be used as a prototyping device for real-time applications due to its speed and hardware-based running. Because according to the test results, FPGA has accelerated the calculations up to 1000 times.

Keywords Integrated circuit · Embedded systems · Particle Swarm optimization · Inverse kinematics · Robotics

1 Introduction

It is clear that robots provide great convenience to us by entering different areas of our lives in different ways. Robots have been the most important factor in the industry becoming huge. Today, there are many industrial robots, large and small, used in fixed, autonomous, serial and parallel. Of course, it is inevitable that such a structure which is at the forefront of the industry will be the center of attention of the research world [1]. Especially the control, movements and structure of robots have been examined by many researchers in the literature and important results and analyzes have been obtained [2]. The subject of kinematics, especially the inverse kinematics equations and the solution of the problem, has been repeatedly investigated in this field. Because inverse kinematics equations have revealed a non-linear, complex and time-consuming problem [3]. Inverse kinematics is to obtain joint angles from the position information of the end effector of a robot manipulator in the work space [4]. Until 20 years ago, the researchers used numerical, algebraic and geometric methods known as conventional methods to obtain joint angles from the position of the end effector. However, these methods were insufficient due to complexity limitations in solving the inverse kinematics problem with the increasing degrees of freedom of the robots [5]. Later, even today, intelligent optimization techniques have taken their place in the literature as the most effective methods in solving this problem [6]. This process has started with artificial neural networks and evolutionary algorithms; for a long time this techniques have produced effective results in solving the inverse kinematics problem [7]. However, over time, the calculation times obtained with these techniques have become insufficient for use in real robots [8]. For this reason, researchers preferred heuristic algorithms [9] because of their advantages such as quick convergence to solution, producing effective solutions and reasonable time of solution [10]. Techniques such as particle swarm optimization [11, 12], artificial bee colony [13], firefly algorithm [14], wolf colony algorithm are the solution methods that come to the fore in this case. In the last few years, the current situation has

✉ Serkan Dereli
dereli@sakarya.edu.tr

[1] Computer Technology Department, Sakarya University of Applied Sciences, Sakarya, Turkey

[2] Electrical-Electronics Engineering, Sakarya University of Applied Sciences, Sakarya, Turkey

shifted from a software-based solution to a hardware-based solution in parallel with the development of technology.

FPGAs are the manifestation of the idea of designing digital circuits since the 1970s. Today, FPGAs are used as integrated circuits that accelerate systems by providing high speed and performance, and whose design can be changed later [15]. Therefore, FPGA-based solution approaches proposed especially for high-performance situations have attracted the attention of the researchers [16]. After performing the path planning of a mobile robot named Pioneer 3-DX with FPGA-based genetic algorithm, Tuncer and Yıldırım performed the field test. The authors, who focused on the problem of obtaining the solution time of the genetic algorithm for very long periods, reduced this time to real-time study levels in their studies. With their own FPGA designs, they have achieved an incredible improvement by reducing the solution time of the genetic algorithm by 90% [17]. Allaire et *al.* have designed a high-performance hardware-based genetic algorithm to enable the unmanned aircraft to follow the shortest route. When they compared the results of the software-based GA and the hardware-based GA, they clearly showed that there was a difference of 10,000 times [18]. Alabdo et *al*. have developed a parallel FPGA-based architecture for the control of a robotic system. This parallel design, which operates in an image-based dynamic structure, has greatly reduced the controller's response time and increased sensitivity [19]. Irgens et *al.* have implemented the Viola-Jones face recognition algorithm, which is successful in observing images in a video and finding the desired people, with a FPGA-based system. They have also demonstrated the success of their systems by verifying their applications on a true FPGA device [20]. In another study, Dereli proposed a hardware-based solution for the forward kinematics calculation of a 7-joint robot manipulator. In addition, by running this hardware both in series and parallel in accordance with the structure of FPGAs, it has ensured that the result is obtained in a much shorter time [21].

This study has been focused on optimizing solution times so that inverse kinematic solutions of articulated robots can be used in real-time studies with the help of heuristic algorithms. For this purpose, the PSO algorithm was designed and tested in accordance with the FPGA architecture to solve the inverse kinematics problem of a complex 7-joint robot manipulator. The proposed design was synthesized and tested for performance on the Nexys 4 DDR device and implemented in a true 7-DOF robot manipulator. The results were analyzed by comparing them with the software results of different swarm-based heuristic algorithms.

The next part of the study is organized as follows: In the second section, advanced kinematic equations for the robot manipulator to be used in the study are obtained. In addition, details of the FPGA-based design encoded in VHDL language are explained. In the third section, the results are divided into three sub-sections and analyzed in depth.

2 Materials and Methods

2.1 Kinematic Analysis of a 7-Dof Robot Manipulator

Today, researchers use 7-joint robot manipulators in their work. Because they need complex problems to show the effect of the algorithms they have developed for the solution [22]. In addition, these manipulators have the advantage of having more than one joint, avoiding obstacles and having more than one solution to perform a task because they have too many joints [23].

The robot manipulator used in this study, whose joint structure is shown in Fig. 1, is 7-jointed and includes nine Dynamixel AX-12A servo motors. The second joint torque, which is the main axis of the robot manipulator, has supported with a double motor because of insufficient torque. In addition, an additional servo motor has been used for the end element mounted on this robot manipulator called Sungur 375.

Kinematic performance in robot manipulators is directly related to the structure of the robot and helps the manipulator to operate flexibly and efficiently in the work space [24]. In robot manipulators, kinematics reveals the relationship between joints and the position of the end effector in the work space [25]. In this study, kinematics analyzes have been obtained using Denavit-Hartenberg parameters and are shown in Table 1.

The transformation matrix of each joint of the robot manipulator has been obtained with the help of the general transformation matrix given in Eq. 1. The forward direction kinematics has been obtained by performing the process given in Eq. 2, that is, multiplying the transformation matrix of each joint. In this study, the orientation parameters are omitted because the x, y and z positions of the end element are desired.

$$ {}_{i-1}{}^{i}T = \begin{bmatrix} \cos\theta\mathrm{i} & -\cos\alpha i.\sin\theta\mathrm{i} & \sin\alpha i.\sin\theta\mathrm{i} & ai.\cos\theta\mathrm{i} \\ \sin\theta\mathrm{i} & \cos i.\cos\theta\mathrm{i} & -\cos\theta\mathrm{i}.\sin\mathrm{i} & ai.\sin\theta\mathrm{i} \\ 0 & \sin\alpha i & \cos\alpha i & di \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1) $$

In Eq. 1, a, alpha, d and theta represent DH parameters and a new value assignment must be made for each joint. These values are obtained based on the x, y and z axes where the joints are located. a is the length of each joint, ie the offset value of the joint with the center of the previous joint, based on the x centers of the joints; alpha is the angle of change of a joint relative to the z-axis with the previous joint; d is the z-axis offset of the joints; Finally, theta angle is the x-axis

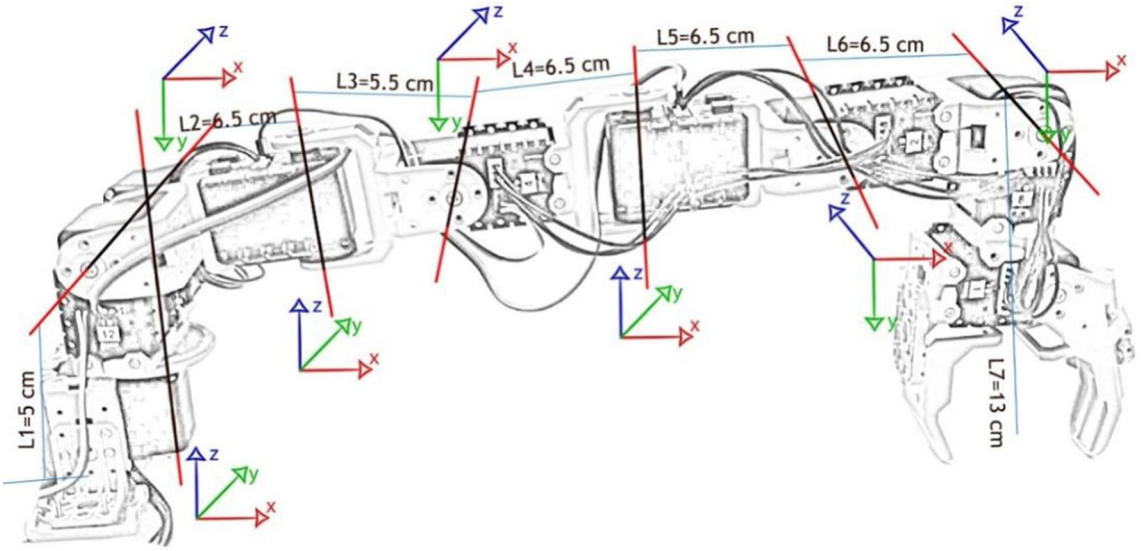

Fig. 1 The structure of 7-dof robot manipulator named Sungur 375

Table 1 DH parameters of 7-dof robot manipulator

i	a_i (cm)	α_i (°)	d_i(cm)	Θ_i (°)(Range)
1	0	−90	$L_1 = 5$	$-90 < \Theta_1 < 90$
2	$L_2 = 6{,}5$	−90	0	$-180 < \Theta_2 < 0$
3	$L_3 = 5{,}5$	−90	0	$-90 < \Theta_3 < 90$
4	$L_4 = 6{,}5$	−90	0	$-90 < \Theta_4 < 90$
5	$L_5 = 6{,}5$	90	0	$-90 < \Theta_5 < 90$
6	$L_6 = 6{,}5$	0	0	$-90 < \Theta_6 < 90$
7	$L_7 = 13$	0	0	$-90 < Q_7 < 90$

centered angle value.

$$A_{\text{End-Effector}} = {}_0{}^7T = {}_0{}^1T.{}_1{}^2T.{}_2{}^3T.{}_3{}^4T.{}_4{}^5T.{}_5{}^6T.{}_6{}^7T = \begin{bmatrix} n_x & s_x & a_x & P_x \\ n_y & s_y & a_y & P_y \\ n_z & s_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\begin{aligned} p_x = {} & L_2c\theta_1c\theta_2 - L_5s\theta_5\left(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3\right) + L_3s\theta_1s\theta_3 \\ & + L_5c\theta_5\left(c\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) + c\theta_1s\theta_2s\theta_4\right) \\ & - L_6s\theta_6\left(s\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) - c\theta_1c\theta_4s\theta_2\right) \\ & - L_7c\theta_7\left(s\theta_6\left(s\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) - c\theta_1c\theta_4s\theta_2\right)\right. \\ & - c\theta_6\left(c\theta_5\left(c\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) + c\theta_1s\theta_2s\theta_4\right)\right. \\ & \left. - s\theta_5\left(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3\right)\right)\Big) \\ & + L_6c\theta_6\left(c\theta_5\left(c\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) + c\theta_1s\theta_2s\theta_4\right)\right. \\ & \left. - s\theta_5\left(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3\right)\right) \\ & - L_7s\theta_7\left(c\theta_6\left(s\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) - c\theta_1c\theta_4s\theta_2\right)\right. \\ & + s\theta_6\left(c\theta_5\left(c\theta_4\left(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3\right) + c\theta_1s\theta_2s\theta_4\right)\right. \\ & \left. \left. - s\theta_5\left(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3\right)\right)\right) \\ & + L_4c\theta_4(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3) \\ & + L_3c\theta_1c\theta_2c\theta_3 + L_4c\theta_1s\theta_2s\theta_4 \end{aligned} \tag{3}$$

$$\begin{aligned} p_y = {} & L_5s\theta_5\left(c\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3\right) \\ & + L_7s\theta_7\left(c\theta_6\left(s\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) + c\theta_4s\theta_1s\theta_2\right)\right. \\ & + s\theta_6\left(c\theta_5\left(c\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) - s\theta_1s\theta_2s\theta_4\right)\right. \\ & \left. \left. - s\theta_5\left(c\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3\right)\right)\right) + L_2s\theta_2s\theta_1 - L_3c\theta_1s\theta_3 \\ & - L_5c\theta_5\left(c\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) - s\theta_1s\theta_2s\theta_4\right) \\ & + L_6s\theta_6(s\theta_4(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1) + c\theta_4s\theta_1s\theta_2) \\ & - L_4c\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) \\ & - L_6c\theta_6\left(c\theta_5\left(c\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right)\right.\right. \\ & \left. - s\theta_1s\theta_2s\theta_4\right) - s\theta_5\left(s\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3\right)\Big) \\ & + L_7c\theta_7\left(s\theta_6\left(s\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) + c\theta_4s\theta_1s\theta_2\right)\right. \\ & - c\theta_6\left(c\theta_5\left(c\theta_4\left(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1\right) - s\theta_1s\theta_2s\theta_4\right)\right. \\ & \left. \left. - s\theta_5\left(c\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3\right)\right)\right) \\ & + L_3c\theta_2c\theta_3s\theta_1 + L_4s\theta_1s\theta_2s\theta_4 \end{aligned} \tag{4}$$

$$
\begin{aligned}
p_z = {} & L_1 - L_2 s\theta_2 + L_6 s\theta_6 \left(c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4\right) \\
& - L_7 s\theta_7 \left(s\theta_6 \left(c\theta_5 \left(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2\right) - s\theta_2 s\theta_3 s\theta_5\right)\right. \\
& \left. - c\theta_6 \left(c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4 + c\theta_3 s\theta_2 s\theta_4\right)\right) \\
& - L_3 c\theta_3 s\theta_2 + L_4 c\theta_2 s\theta_4 \\
& + L_6 c\theta_6 \left(c\theta_5 \left(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2\right) - s\theta_2 s\theta_3 s\theta_5\right) \\
& + L_5 c\theta_5 \left(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2\right) \\
& + L_7 c\theta_7 (c\theta_6 (c\theta_5 (c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5) \\
& \qquad + s\theta_6 (c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4)) \\
& - L_4 c\theta_3 c\theta_4 s\theta_2 - L_5 s\theta_2 s\theta_3 s\theta_5
\end{aligned} \tag{5}
$$

2.2 Model of Position Error and Fitness Function

One of the main objectives of this study is to calculate the optimal joint angles that will lead the robot manipulator end effector to the predetermined point using forward kinematics equations. The joint angles obtained direct the robot manipulator to a certain position in the work space. Position error is the distance between the actual position of the end effector and the desired position of the end effector, and in this paper, Eq. 6, known as the Euclidean distance equation in the literature, is used to calculate the position error.

$$
PositionError = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{6}
$$

Figure 2 is the explicitly stated position error for this study. "P2" represents the position that the end effector has to reach and "P1" indicates the actual position.

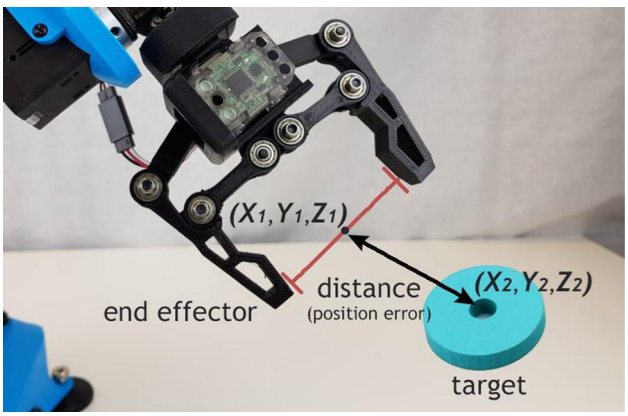

Fig. 2 Illustration of position error

2.3 FPGA-Based Particle Swarm Optimization (IC-PSO)

This technique, first used by Kennedy and Eberhart, is a powerful search algorithm inspired by swarm of birds and fish [26]. As shown in Fig. 3, the particles in the swarm are always moving toward the target. It achieves this with the new position calculated according to the velocity of the particle at each iteration. If the calculated new position of any particle is worse than the old position, the new position is ignored and the old position is considered the new position.

The PSO flowchart used in this study is shown in Fig. 4. As with all heuristic algorithms, the processing cycle in this algorithm starts with random numbers. After generation of random numbers, it is checked whether the joint angles are within the ranges specified in Table 1. Afterward, x, y and z position in the work space is calculated by performing forward kinematics calculations of the particle with the obtained joint angles. After calculating the position error of the particle, the local best value and the global best value are obtained. If the stop criterion is not reached, the velocity and angle values of the particles are updated, and continue from the next iteration.

The hardware-based high-performance particle swarm optimization (PSO) block diagram is shown in Fig. 5. This figure is also known as the state diagram in digital systems, and in each case one or more sub-modules run sequentially as hardware. The control unit is an indispensable unit for the circuit to run in the correct order. Random angles (particles), local best values, general best values and velocity values of particles are stored in RAM, the memory unit. Some FPGA devices have external block RAM, but not many. In these devices, data is stored on flip flops that actually act as a single cell of RAM [27]. "Target x, y, z" is the coordinates of the point of the robot manipulator must reach in the work space and it is transferred to the system from outside. In the implementation of this study, the coordinates are transferred from the computer. The block diagram in Fig. 5 is executed according to the flowchart of the algorithm of Fig. 4.

All modules are designed using the VHDL hardware description language, and the system is first tested in simulation and then implemented with FPGA. The red arrows shown in Fig. 5 not only express the mechanism that updates the particle velocity and position, but also initiates a new iteration in the algorithm. When the design is considered in terms of the size of the integrated systems, it can be considered as a VLSI design with a total of 12 sub-modules. These sub-modules are explained in detail below at this stage.

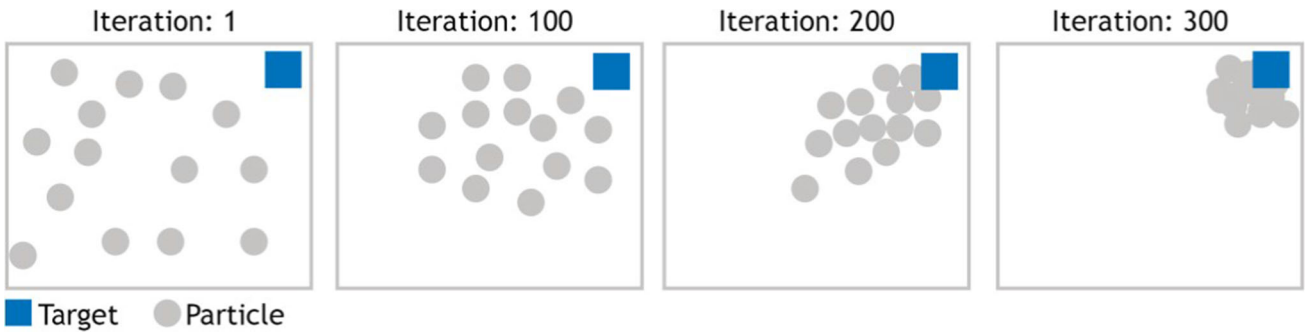

Fig. 3 Motion of particles toward the target in PSO algorithm

Fig. 4 FPGA-based PSO flowchart

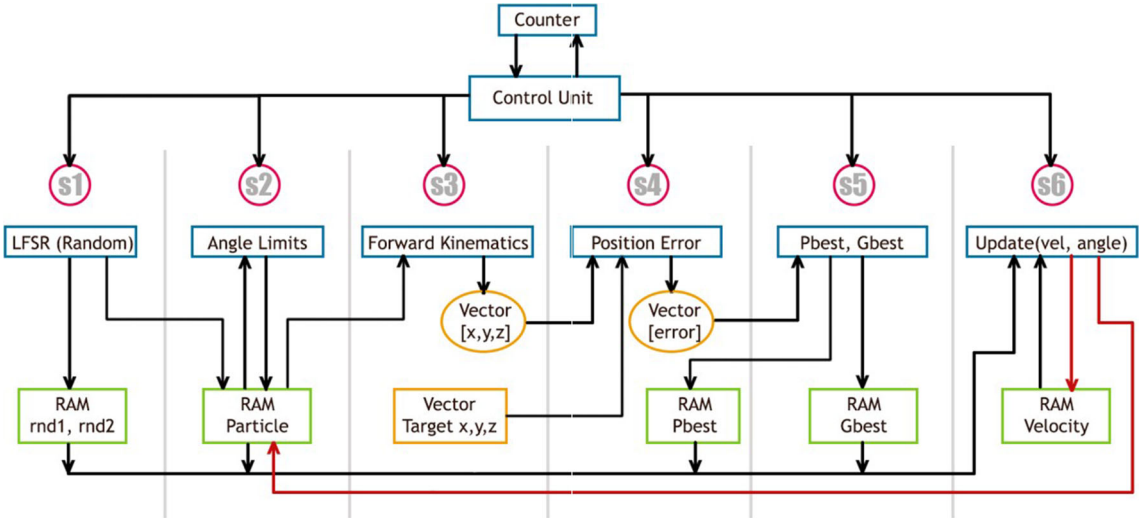

Fig. 5 FPGA-based IC-PSO block diagram

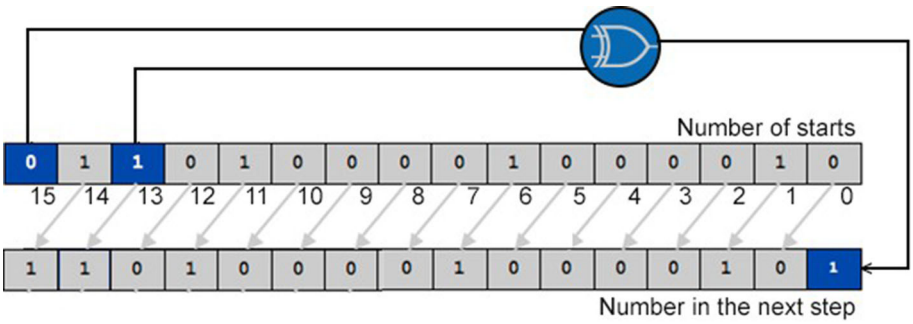

Fig. 6 LFSR random number generator used in this study

2.3.1 LFSR (Linear Feedback Shift Register)

All random values used in the system were generated by the LFSR technique, which is used as an effective pseudo-random number generator in the literature. As it is known, LFSR, which is frequently preferred in the key generation phase with a uniform distribution in encryption algorithms, was preferred as 16-bit in this study [28]. The LFSR used in this article performs the XOR operation between the thirteenth and fifteenth bits to generate 16-bit random numbers. The 16-bit LFSR technique illustrated in Fig. 6 was used in this study.

2.3.2 Angle Limits

The joints in the robot manipulator perform positioning by taking angle values at certain intervals. Exceeding this limit causes the manipulator to be in the wrong position or to move to a position other than the desired position. This sub-module is also used to prevent the joints from colliding by taking an angle value in the wrong range. For this reason, this stage is used in all heuristic algorithms and ensures that the angles remain within certain value ranges that appear in Table 1. As shown in this table, it is clear that all angle values except the second joint are the same.

$$\text{For Second Angle} \quad \text{Angle Value} = \begin{cases} -180, & \text{angle} < -180 \\ 0, & \text{angle} > 0 \end{cases} \qquad \text{For Other Angles} \quad \text{Angle Value} = \begin{cases} -90, & \text{angle} < -90 \\ 90, & \text{angle} > 90 \end{cases} \tag{7}$$

2.3.3 Forward Kinematics Calculation

In this design, which is used as a sub-logical module, seven joint angles are taken and the position of the manipulator's end effector on the x, y and z axes is obtained through forward kinematics equations. In this module, trigonometric calculations are performed with a 16-bit CORDIC algorithm, while the results are generated as 32-bit floating numbers. The CORDIC (Coordinate Rotation Digital Computer) algorithm is based on capturing the desired angle value by rotating a vector around the origin, as the name suggests. Also, since the CORDIC algorithm does not contain multiplication or divisions, its cost to the computer is extremely low [29].

As shown in Fig. 7, two IP cores, Cordic and Floating Point, are used in the logical sub-module. Cordic IP is an IP Core that produces the sine and cosine values of 16-bit input numbers as 16-bit + 16-bit, i.e., 32-bit in total. Floating Point IP is used to convert 16-bit fixed numbers to 32-bit decimal number. Thus, by performing forward kinematics calculation, the positions where the angles guide the end element are obtained.

As seen in Fig. 7, this sub-design consists of four finite states, "s1, s2, s3 and s4." In the first finite state (s1), the joint angles are transferred to a vector and made ready for process. In the second finite state (s2), the joint angles present in the vector are transferred to the CORDIC algorithm in order and sine and cosine values are obtained. The angles are now at the stage of converting to decimals, and this is done in the third finite state (s3). In the last state (s4), forward kinematics calculated using the sine and cosine angle values and position information of the end effector is obtained.

2.3.4 Position Error

In the heuristic algorithms, the rate at which the values obtained at the end of iteration approach the target point is questioned in this sub-module. Since heuristic techniques are randomly approaching the target, how close the solution is to the desired point is determined by the position error value. So, in this sub-design, the distance of the end effector of the robot manipulator to the target point is calculated using Euclidean Equation (Eq-6).

2.3.5 Best Values

At this stage of the digital circuit, the process of finding the position closest to the target point in the work space of the robot manipulator is performed. In the previous steps, the position of each particle in the work space was calculated and the particle with the best local value (pbest) information was obtained. In this sub-module, it is questioned whether the particle with the best local value (pbest) has the best value obtained so far (gbest). For this, the "pbest" value and the "gbest" value are compared in terms of position error, and the particle with the small position error continues the algorithm as the best particle. As a result of this comparison process, the new global best value (Gbest) is calculated according to Equation-8.

$$G\text{best}_{\text{new}} = G\text{best}_{\text{old}} - P\text{best} \tag{8}$$

2.3.6 Update Velocity and Position

According to the PSO algorithm, this stage is one of the important steps that the particles use to reach the desired position. This logical sub-module also consists of five finite state (s1, s2, s3, s4, s5) as shown in Fig. 8 and the transition to each stage takes place via the control unit as in other logical circuits. The most prominent feature of this step is that all operations are performed as 32-bit floating numbers. For this reason, all the numbers in the circuit were first expanded to 32-bit and then converted to a floating number. The angle values transferred to the output ports are 16-bit and the speed values are 32-bit.

IW (Inertia Weight) is known as the velocity control parameter in particle swarm optimization algorithm and the value of this parameter can be changed by many methods [30]. Because the IW parameter enables the particles to move toward the specified target and increases the stability of the algorithm [31]. In this study, although a fixed IW value is used, a separate step is added to the logical design as it appears in the block diagram. In this way, other IW techniques used in the literature can be easily integrated into the system.

In the s3 and s4 stages of the design, the particles are brought closer to the target according to the PSO algorithm rule. For this, the particle velocity at the s3 stage and the position of the particle at the s4 stage, that is, the new joint angle, are calculated. Equation-9 is used for velocity calculations and Equation-10 is used for position, i.e., joint angle calculation.

$$v_{id} = v_{id} + c_1.r_1.(p_{\text{best}} - x_{id}) + c_2.r_2.(g_{\text{best}} - x_{id}) \tag{9}$$

$$x_{id} = x_{id} + v_{id} \tag{10}$$

In Eqs. 9 and 10; "d" is the dimension of the problem; "i" is the number of particles; "c1" and "c2" personal best and global best weights; "r1" and "r2" represent random numbers in the range [0–1]. "Pbest" is the distance of the location of each particle to the food source, i.e., the optimum solution. "Gbest" is the closest distance the swarm has achieved according to the food source during iteration.

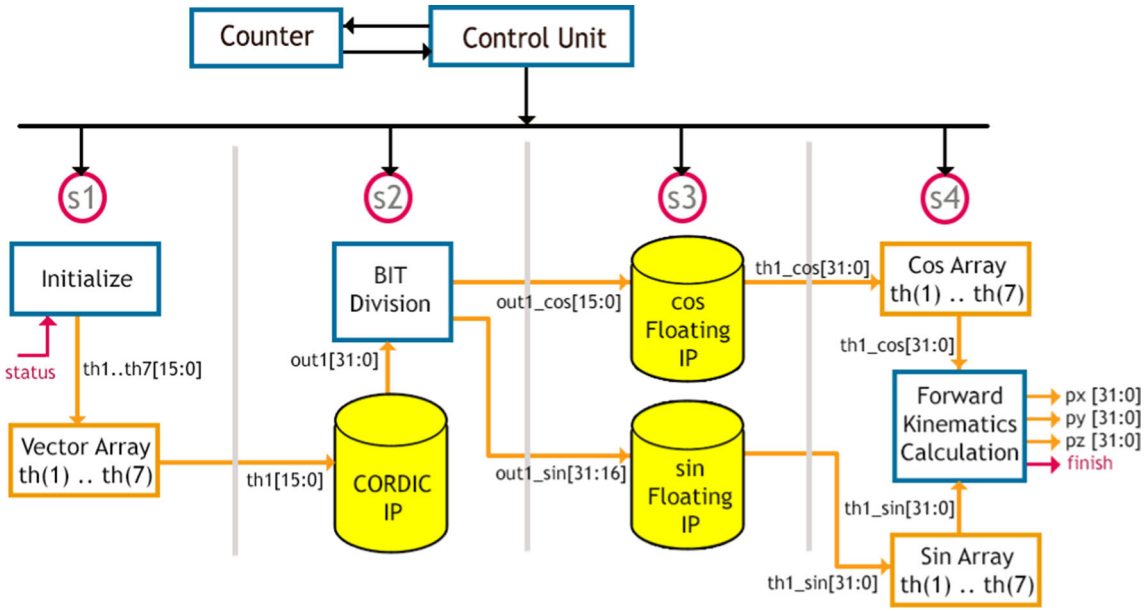

Fig. 7 Forward kinematics block schema

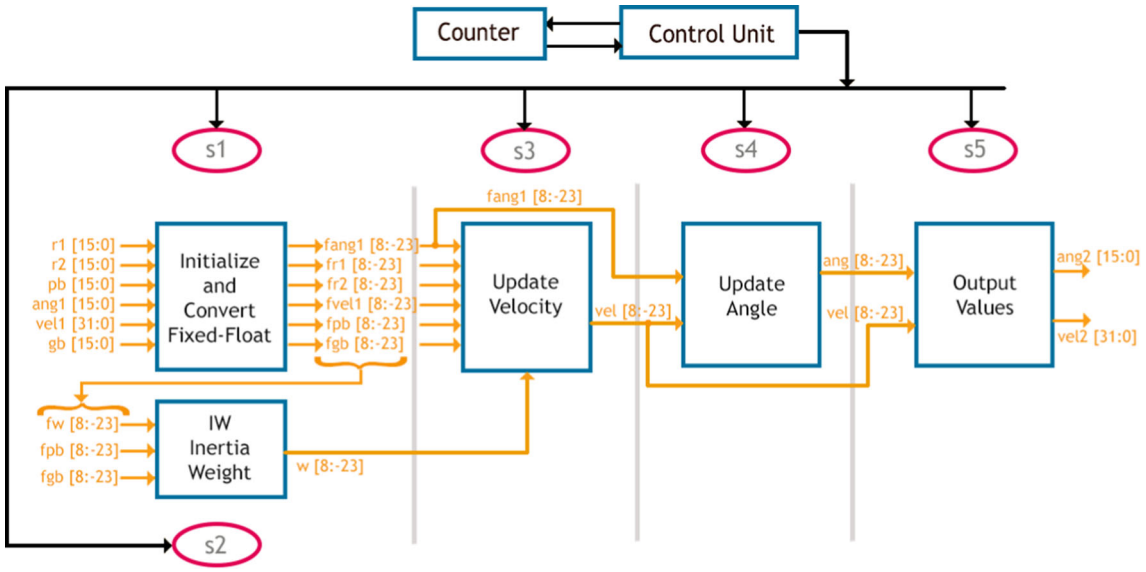

Fig. 8 Block schema of the update position

3 Results

In this section, simulation and implementation results of hardware-based particle swarm optimization algorithm with FPGA are examined. This digital circuit, which can be synthesized, has been implemented in the Vivado interface 2017.4 using the VHDL hardware design language.

3.1 Simulation Results

The hardware-based PSO is designed using the VHDL language in the Vivado interface. The digital circuit design has been tested separately with 30, 50 and 100 populations and the results have been demonstrated. Also, as a result of the joint angles obtained, the orientation of the robot manipulator has been demonstrated through the Roboanalyzer simulation software [32]. Parameters and values used in designs are as follows:

- Iteration number: 50
- $c_1 = 1.4$
- $c_2 = 1.4$
- $w = 0.7$

Table 2 Initial values of random numbers

Type	Rnd_1	Rnd_2	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
Hex	c526	8207	1921	f7a0	02ca	fd36	0b2b	f4d5	0860
Fixed-Point (Radian)	0.7700	0.5078	0.7853	−0.5983	0.0871	−0.7728	0.3490	−0.5109	0.2617
Degree	–	–	45.0172	−34.2975	4.9947	−44.3006	20.0064	−29.2873	15.0019

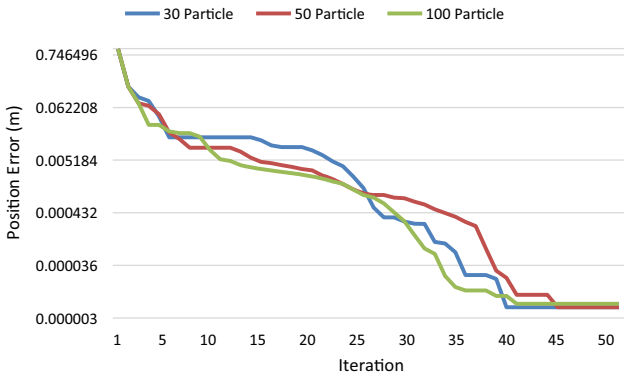

Fig. 9 Position error of different particles-iteration graph

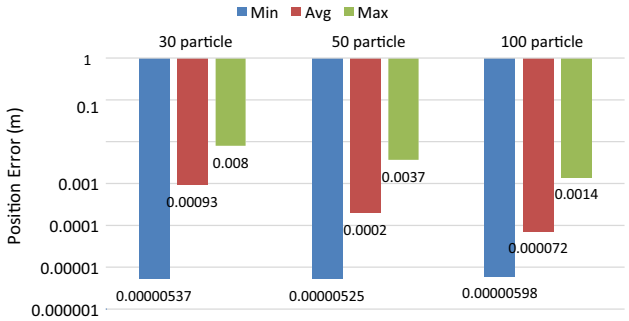

Fig. 10 Minimum, maximum and average position error values of each swarm

In the designs, LFSR is used for random numbers and these random values to be used in each iteration are initially generated. Similarly, joint angles have been randomly generated at the initial of the PSO algorithm to assign 16-bit angle values. In the conversion of this 16-bit to radian, MSB bit represents the sign bit, the next 2-bit integer, and the remaining 13-bit represents the fraction part. Rnd_1 and Rnd_2, which take a value between [0–1], show the random numbers used when updating the velocity of the particles while the others (θ_1–θ_7) show the values of the joint angles. The initial values of randomness in both random numbers and joint angles are as follows:

The results were compared in terms of position error and time to reach the solution. Since the time in FPGA is directly related to the frequency value of the chip on the board, the number of clock pulses in the runs is also indicated next to the time. Because the Nexys 4 DDR device used in this study has a clock frequency of 450 MHz, the designs have been tested with 3 ns clock pulses. If the device used has a higher frequency, the operating time will of course be reduced.

The design consists of seven parts: 'Random Numbers', 'Random Angles', 'Angle Limits', Forward Kinematics', 'Position Error', 'Best Values' and 'Updating Angle and Velocity. When analyzing the results, the times used for each section as well as the number of clock pulses are particularly indicated. Because in FPGAs, the factor that actually constitutes the time is the clock pulse in which the circuit runs [33] (Table 2).

Figure 9 shows the best position error values obtained when the hardware-based PSO circuit is performed 50 iterations with 30, 50 and 100 particles. The best values of each swarm reaches between 50 iterations are extremely close to each other and are at 10^{-6}. Even when the iterations in which the solution values are obtained are examined, it is clear that there is a similarity.

Figure 10 shows the average, best and worst values of 50 different tests performed with hardware-based PSO. Therefore, it appears that there is a 1000 times difference between the best value and the worst value. However, it is obvious that even the worst values are quite close to the best values of the software-based algorithm. The main subject of this study is to bring the solution times closer to the real time.

Table 3 shows the time at which the best position error values obtained according to the number of particles reached a solution and the number of clock pulses. Although the position error values are very close to each other, quite different values emerge in terms of solution time. Of course, since these values are in milliseconds, they seem to be small differences in the workings of the mechanisms, but double the differences in science. When we look at the number of clock pulses, it is clear that this difference is double.

Figure 11 shows the solution times obtained with each particle as a minimum, maximum and average value after 50 tests. In fact, these times are directly related to the number of iterations performed. Because as the number of iterations increases, the number of clock pulses increases. Therefore, the solution time increases.

In terms of position error and calculation time, the best values and average values obtained are shown in Table 4. In this

Table 3 Comparative results of best values obtained by number of particles

Swarm Number	PositionError (m)	Solution Time (ms) (clk*3 ns)	Solution Iteration	Number of clock pulse (clk)
30	5.37e-06	1.423	38	474,285
50	5.25e-06	2.678	43	876,207
100	5.98e-06	4.867	39	1,622,170

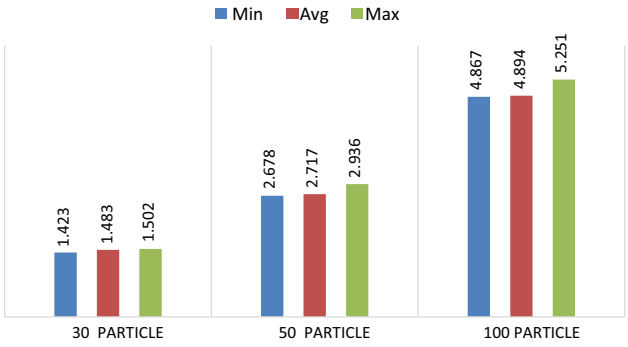

Fig. 11 Minimum, maximum and average solution time values of each swarm

paper, the number of clock pulses is calculated by multiplying by 3 ns, because FPGA device is used with 450 MHz frequency. Looking at the table, the number of clock pulses has increased as the number of particles increases, thus increasing the calculation time.

Table 5 shows the comparison of the software-coded conventional particle swarm optimization, artificial bee colony, firefly algorithm and quantum particle swarm optimization with hardware designed PSO in terms of positional error and calculation time. Obviously, the IC-PSO showed a high level of performance in terms of computation time. The proposed FPGA-based method appears to have 165 times better time than even the quantum PSO with the best computing time. Also, this proposed method has the best values after quantum PSO in terms of position error.

3.2 Implementation of the System

The block diagram of the system created for this purpose can be seen in Fig. 12. In this figure, the FPGA device calculates seven joint angles from the x, y, and z coordinates that the end effector of the manipulator must reach, with the PSO algorithm. The results are transferred to the computer environment via the RS232 interface and from there to the CM-530 control card. The control card directs the joints to these angles according to the cubic trajectory planning, allowing the end effector to reach the desired point. In this study, the main focus is on designing the algorithm at the RTL level. Therefore, the FPGA device is not directly connected to the robot manipulator. Because this stage also includes connecting a microprocessor to the system, transferring the obtained angle information to the pins first and then to the motors via the microprocessor with appropriate resolution. Therefore, this process is considered to be the subject of

Table 4 Comparative results of best values obtained by number of particles

Particle	Position Error (m)	Solution Time (ms) (Clock*3 ns)	Solution Iteration	Solution Clock
Number	Avg./Min	Avg./Min	Avg./Min	Pulse Number
30	9.3e-04/5.37e-06	1.483/1.423	43/38	490,638/474285
50	2.0e-04/5.25e-06	2.717/2.678	45/43	886,146/876207
100	7.26e-05/5.98e-06	4.894/4.867	41/39	1,627,891/1622170

Table 5 Comparison of software values of algorithms with FPGA-based PSO

Algorithm	Swarm Size	Max. Iteration	Position Error (m)	Solution Time (ms)	Speed Improvement
PSO	300	500	6.71e-03	449.8	320 x
ABC	100	500	5.47e-04	444.1	317 x
Firefly	50	500	6.53e-05	920.4	657 x
Quantum PSO	150	500	2.77e-17	231.9	165 x
IC-PSO	30	50	5.37e-06	1.483	–

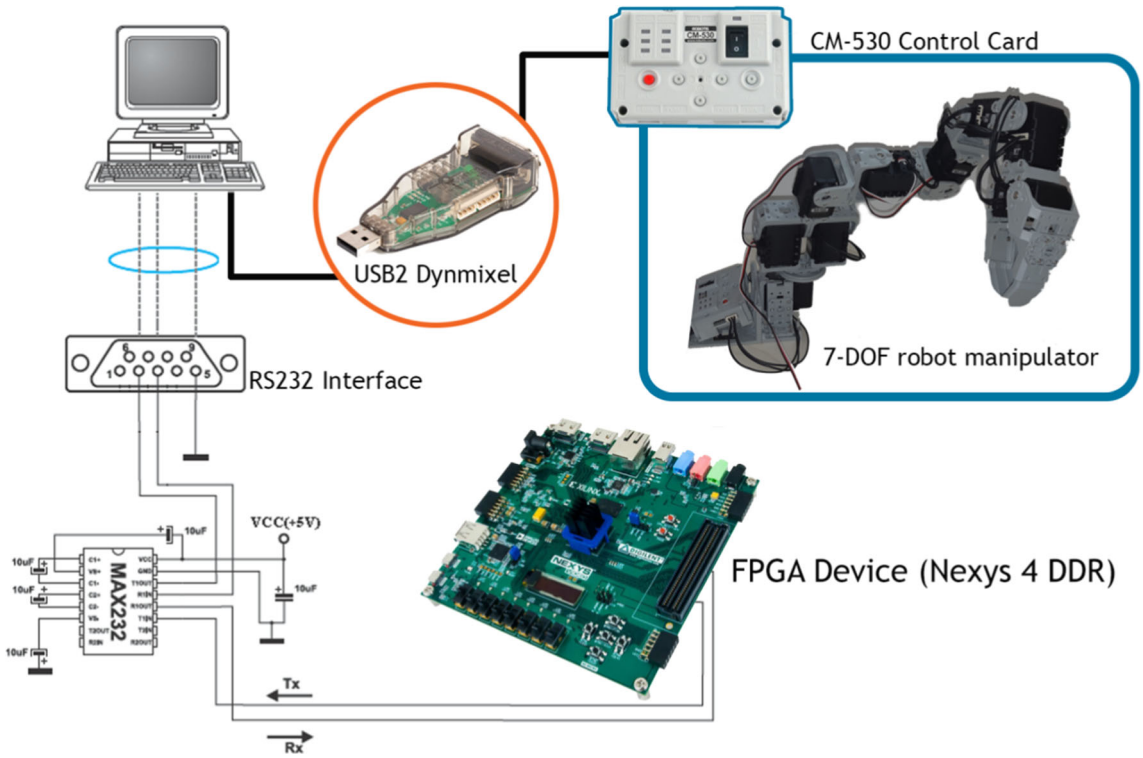

Fig. 12 Connection diagram of the system used

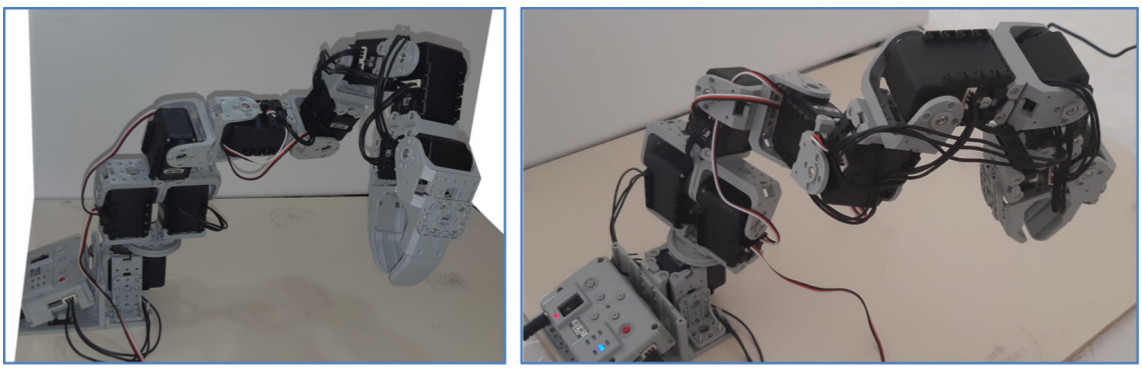

Fig. 13 Manual (Left) and calculated (Right) position of the robot manipulator

another study, since it reveals a situation that is comprehensive and different from the focus of this study.

In Fig. 13, an example image of the position change performed by the robot manipulator is illustrated. In this figure, the image (1) indicates the point that the end effector is desired to reach, while the image (2) indicates the orientation formed by the joint angles calculated with the hardware-based PSO. As a result, although the joint angles were different, the end effector was oriented to the same location.

Figure 14 shows the intermediate values of the angles obtained by the cubic trajectory planning of the joint angles that bring the robot manipulator from its initial position to the desired target position. In the planning made, it is foreseen that the joint angles will reach their latest values in a 1 s time frame. In this context, each joint angle takes 20 different intermediate values and the robot manipulator is positioned at the desired point.

In Fig. 15, sequential positions of a task performed by the robot manipulator are shown. In this task, the robot manipulator takes positions (2), (3), (4), (5) and (6), sequentially, to move the object visible in image (1) to the target. The red lines on the images indicate the position that the robot manipulator will take next.

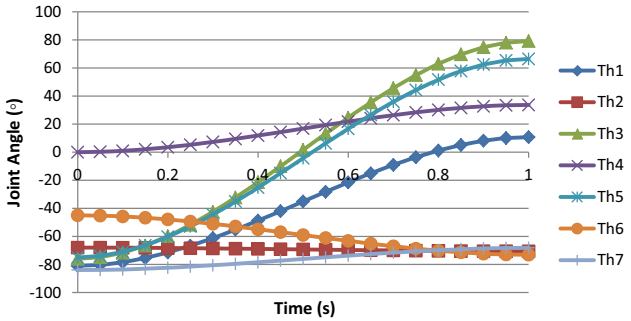

Fig. 14 Cubic trajectory planning of the robot manipulator

3.3 Discussion and Analysis

This study aimed to reduce the solution times by realizing the heuristic algorithms, whose calculation times are very long and which cannot be used in real-time systems, with FPGA. However, when looking at the literature comprehensively, it is clear that heuristic algorithms are used to reduce the position error of the manipulator. Because, all of the heuristic algorithms achieve inverse kinematics solution in almost similar times. Table 6 shows the current literature on the inverse kinematics solution performed with heuristic algorithms in this sense. Studies have been examined in terms of both position error (m) and solution resolution time (s). It can be clearly seen in this table that although the position error value has been reduced to the desired levels, the solution time has not been reduced to real-time operating levels. This makes it difficult to use heuristic algorithms in real-time applications. As a matter of fact, this is the process of lifting or releasing a load by directing the end effector of the robot manipulator from a point to the desired position. Therefore, in real applications, this process itself should take place in a time interval of [0–1] seconds. However, researches in the literature with heuristic algorithms have obtained the solution of the inverse kinematics problem, which is only part of the process, in the interval of [0–1] seconds. This is why it is essential to reduce this time to reasonable times that can be used in real-time applications, which is the focus of this study.

This study reveals a real application for the use of heuristic algorithms in real-time applications thanks to FPGA. For this purpose, firstly, priority has been given to the literature studies where FPGA is used in real applications. The results obtained in the literature studies have been the primary motivation for this study. The fact that there are few studies and that FPGAs are preferred for real-time applications is an important parameter in terms of demonstrating the importance of this study.

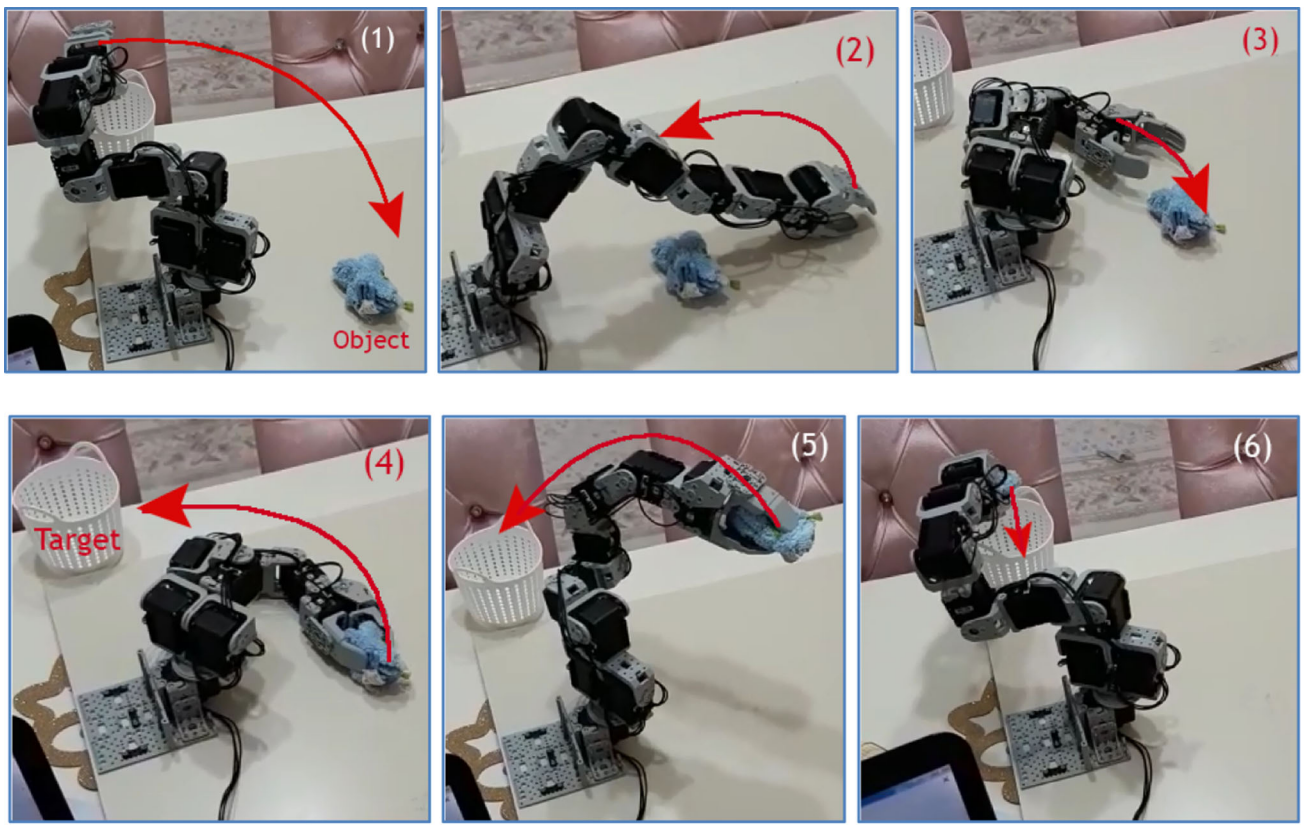

Fig. 15 Exemplary task performed by the robot manipulator

Table 6 Comparison with some studies in the literature

Research	Robot Arm	Reference Technique	Technique Compared	Comparison Type
Ayyıldız and Çetinkaya [8]	4-DOF	PSO	GA	
		7.39e-06	4.74e-04	Position Error (m)
		4.15	16.9	Solution Time (s)
Dereli and Köker [11]	7-DOF	IW-PSO	PSO	
		3.64e-03	9.13e-03	Position Error (m)
		1.2	1.9	Solution Time (s)
Rokbani et al. [14]	3-DOF	IK-FA	Cyclic Coordinate Descent (CCD)	
		10e-04	10e-04	Position Error (m)
		2.19e-03	0.4329	Solution Time (s)
El-Sherbiny et al. [34]	5-DOF	Adaptive Neuro-Fuzzy Inference System (ANFIS)	GA	
		5.426e-03	7.64e-04	Position Error (m)
		0.0308	83.1239	Solution Time (s)
Küçük and Bingül [35]	6-DOF	NIKA (News Inverse Kinematics Algorithm)	Newton–Raphson	
		1.02e-04	0.2266	Position Error (m)
		0.327456	0.331443	Solution Time (s)
Momani et al. [36]	3-DOF	Continuous GA	Conventional GA	
		2.10	5.25	Convergence Speed (s)
Wichapong et al. [37]	4-DOF	DE (Differential Evolution)	ABC (Artificial Bee Colony)	
		5.06e-06	2.45e-03	Position Error (m)
Lopez et al. [38]	5-DOF	Soft Computing Aproach	Traditional Method	
		5.08e-04	6.77e-03	Position Error (m)
		0.7969	1.717	Solution Time (s)
Collinsm and Shen [39]	9-DOF	Bare Bones PSO	Constriction Factor PSO	
		0.00579	0.00232	Position Error (m)
		2.44	4.22	Solution Time (s)
Dalmedico et al. [40]	3-DOF	ANN	–	
		1.12e-03	–	Position Error (m)
Dereli [41]	7-DOF	Improved GWO		
2.78e-17	GWO			
1.159e-04				
Position Error (m)				
Dereli [42]	7-DOF	Improved WOA		
2.96e-15	2.49e-04			
Position Error (m)				

For example, Lee et al. [43] carried out an FPGA-based real-time study to avoid delays and speed up the system in bidirectional request-response data communication. In their study, they compared the systems developed with GPU and CPU to reveal the contribution of the FPGA factor to the system. As a result, they found that the FPGA-based system operates 50% faster than the GPU-based system and 75% faster than the CPU-based system.

Another study is as follows: Li et al. [44] have designed the FPGA design to further develop the convolution neural network (CNN) model in terms of speed and energy efficiency compared to the GPU. Apart from the FPGA design, it is an important element that the proposed architecture has parallel operation and optimization features. According to the results of the experiment, it is stated that the proposed architecture has 7.5 × faster and 75 × more energy efficiency than GPU.

Karakuzu et al. [45] have developed a neuro-fuzzy system that performs meta-heuristic learning through the particle swarm optimization technique with FPGA. The most obvious feature of the proposed system is that it reduces the multiplier coefficients and consequently excess memory space. In this way, the proposed system has produced a more effective study than the existing systems.

Li et al. [46] made motion planning with an FPGA-based design for an autonomous mobile robot operating in real time. As the most time-consuming issue in path planning of a mobile robot is collision detection, they have implemented a visual system based design. They proved their effectiveness by experimental test the proposed design on a real robot.

4 Conclusion

In this study, an architecture that produces FPGA-based inverse kinematics solution is proposed in order to use heuristic algorithms in real-time robotic applications. The design architecture used in the study has realized the inverse kinematics solution of the 7-DOF serial manipulator based on the PSO algorithm. Because the proposed method is synthesizable, it has been run on a Nexys 4 DDR device and the results have been tested in real time on a 7-DOF robot manipulator. The experimental test of the design was carried out in two ways. Initially, the robot manipulator has been directed to a single point and the inverse kinematics solution of this point was calculated. Then, in order to analyze the accuracy of the method, the robot manipulator has been directed to 50 different points and the results have been obtained. The findings were analyzed in terms of position error and especially calculation time and compared with the software results such as PSO, ABC, FA and QPSO. The results obtained by the proposed hardware-based method are at least 150 times better in terms of calculation time and 10 to 1000 times better in terms of position error. Thus, it is seen that the FPGA-based designs allow the real-time operation of the systems that involve complex and time-consuming processes. The limitation of this study is that the FPGA-based design is unique only to the robot manipulator used in this article. However, if the FPGA-based design is realized on the basis of Denavit-Hartenberg (DH) parameters, it will become a state that can be used for all robot manipulators. In parallel with this situation, the inverse kinematic solution of any robot can be obtained comparatively by creating one by one IP core in a compact structure for similar heuristic algorithms.

References

1. Dörfler, K.; Dielemans, G.; Lachmayer, L.; Recker, T.; Raatz, A.; Lowke, D.; Gerke, M.: Additive Manufacturing using mobile robots: opportunities and challenges for building construction. Cem. Concr. Res. **158**, 1–13 (2022)
2. Brahmia, B.; Maarouf, S.; Jacqueline, T.; Cristobal, O.; Philippe, S.; Mohammad, H.: Adaptive control of a 7-DOF exoskeleton robot with uncertainties on kinematics and dynamics. Eur. J. Control. **42**, 77–87 (2018)
3. Iliukhin, V.; Mitkovskii, K.; Bizyanova, D.: The modeling of inverse kinematics for 5 DOF manipulator. Proc. Eng. **176**, 498–505 (2017)
4. Kucuk, S.; Bingul, Z.: Inverse kinematics solutions for industrial robot manipulators with offset wrists. Appl. Math. Model **38**, 1983–1999 (2015)
5. El-Sherbiny, A.; El-Hosseini, M.; Haikal, A.: A new ABC variant for solving inverse kinematics problem in 5 DOF robot arm. Appl. Soft Comput. **73**, 24–38 (2018)
6. Zhang, Y.; Zhou, Y.; Zhou, G.; Luo, Q.; Zhu, B.: A Curve Approximation approach using bio-inspired polar coordinate bald eagle search algorithm. Int. J. Comput. Intell. Syst. **15**, 1–25 (2022)
7. Köker, R.: A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. Inf. Sci. **222**, 528–543 (2013)
8. Ayyıldız, M.; Çetinkaya, K.: Comparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-DOF serial robot manipulator. Neural Comput. Appl. **36**, 825–836 (2016)
9. Abainiave, K.; Ali, Y.: Bio-inspired approach for inverse kinematics of 6-DOF robot manipulator with obstacle avoidance. In: International conference on pattern analysis and intelligent systems, 2018
10. Dereli, S.; Köker, R.: A meta-heuristic proposal for inverse kinematics solution of 7-DOF serial robotic manipulator: quantum behaved particle swarm algorithm. Artif. Intell. Rev. **53**, 949–964 (2020)
11. Dereli, S.; Köker, R.: Iw-PSO approach to the inverse kinematics problem solution of a 7-DOF serial robot manipulator. Sigma **36**, 77–85 (2018)
12. Sancaktar, I.; Tuna, B.; Ulutas, M.: Inverse kinematics application on medical robot using adapted PSO method. Eng. Sci. Technolo. An Int. J. **21**, 1006–1010 (2018)
13. Zhang, L.; Xiao, N.: A novel artificial bee colony algorithm for inverse kinematics calculation of 7-DOF serial manipulators. Soft. Comput. **23**, 3269–3277 (2019)
14. Rokbani, N.; Casals, A.; Alimi, A.: IK-FA, a new heuristic inverse kinematics solver using firefly algorithm. In: Computational Intelligence Applications in Modeling and Control, pp. 369–395, 2015.
15. Zeng, K.; Ma, Q.; Wu, J.; Chen, Z.; Shen, T.; Yan, C.: FPGA-based accelerator for object detection: a comprehensive survey. The J. Supercomput. **78**, 14096–14136 (2022)
16. Naru, E.; Saini, H.; Sharma, M.: A recent review on lightweight cryptography in IoT. In: International Conference on I-SMAC, 2017
17. Tuncer, A.; Yıldırım, M.: Design and implementation of a genetic algorithm IP core on an FPGA for path planning of mobile robots. Turk. J. Elec. Eng. Comp. Sci. **24**, 5055–5067 (2016)
18. Allaire, F.; Tarbouchi, M.; Labonté, G.; Fusina, G.: FPGA implementation of genetic algorithm for UAV real-time path planning. J. Intell. Robot. Syst. **54**, 495–510 (2009)
19. Alabdo, A.; Garcia, J.P.; Pomares, J.; Torres, F.: FPGA-based architecture for direct visual control robotic systems. Mechatronics **39**, 204–216 (2016)
20. Irgens, P.; Bader, C.; Lé, T.; Saxena, D.; Ababei, C.: An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm. HardwareX **1**, 68–75 (2017)
21. Dereli, S.: Micro-sized parallel system design proposal for the solution of robotics based engineering problem. Microsyst. Technol. **27**, 4217–4226 (2021)

22. Simas, H.; Di Gregorio, R.: Collision avoidance for redundant 7-DOF robots using a critically damped dynamic approach. Robotics **11**, 1–19 (2022)
23. Chen, X.; Zhao, B.; Wang, Y.; Xu, S.; Gao, X.: Control of a 7-DOF robotic arm system with an SSVEP-based BCI. Int. J. Neural Syst. **28**, 1–15 (2018)
24. Kucuk, S.; Bingul, Z.: Comparative study of performance indices for fundamental robot manipulators. Robot. Autonom. Syst. **54**, 567–573 (2006)
25. Toshani, M.: Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: a Lyapunov-based approach. Robot. Auton. Syst. **62**, 766–781 (2014)
26. Kennedy, J.; Eberhart, R.: Particle Swarm optimization. In: International Conference on Neural Networks, Perth, Australia, 1995.
27. Yang, H.; Zhang, J.; Sun, J.; Yu, L.: Review of advanced FPGA architectures and technologies. J. Electr. (China) **31**, 371–393 (2014)
28. Petrica, L.: FPGA optimized cellular automaton random number generator. J. Parallel Distrib. Comput. **111**, 251–259 (2018)
29. Li, J.; Fang, J.; Li, B.; Zhao, Y.: Study of CORDIC algorithm based on FPGA. In: Chinese Control and Decision Conference (CCDC), 2016
30. Harrison, K.; Engelbrecht, A.; Ombuki-Berman, B.: Inertia weight control strategies for particle swarm optimization. Swarm Intell. **10**, 267–305 (2016)
31. Jiao, B.; Lian, Z.; Gu, X.: A dynamic inertia weight particle swarm optimization algorithm. Chaos Solit. Fract. **37**, 698–705 (2008)
32. Gupta, V.; Chittawadigi, R.; Saha, S.: RoboAnalyzer: robot visualization software for robot technicians. In: Advances in Robotics, 2017
33. Monmasson, E.; Idkhajine, L.; Cirstea, M.; Bahri, I.; Tisan, A.; Naouar, M.W.: FPGAs in industrial control applications. IEEE Trans. Industr. Inf. **28**, 224–243 (2011)
34. El-Sherbiny, A.; Elhosseini, M.A.; Haikal, A.Y.: A comparative study of soft computing methods to solve inverse kinematics problem. Ain Shams Eng. J. **9**, 2535–2548 (2017)
35. Kucuk, S.; Bingül, Z.: Inverse kinematics solutions for industrial robot manipulators with offset wrists. Appl. Math. Model. **38**, 1983–1999 (2014)
36. Momani, S.M.; Hammour, Z.; Alsmadi, O.: Solution of inverse kinematics problem using genetic algorithms. Appl. Math. Inf. Sci **10**, 1–9 (2016)
37. Wichapong, K.; Bureerat, S.; Pholdee, N.: Solving inverse kinematics of robot manipulators by means of meta-heuristic optimisation. IOP Conf. Series: Mater. Sci. Eng. (2018). https://doi.org/10.1088/1757-899X/370/1/012056
38. Lopez-Franco, C.; Hernandez-Barragan, J.; Alanis, A.Y.; Arana-DanieL, N.: A soft computing approach for inverse kinematics of robot manipulator. Eng. Appl. Artif. Intell. **74**, 104–120 (2018)
39. Collinsm, T.J.; Shen, W.M.: Particle swarm optimization for high-DOF inverse kinematics. In: IEEE International Conference on Control, Automation and Robotics, 2017
40. Dalmedico, J.F.; Mendonça,M.; Souza, L.B.; Barros, R.V.P.D.; Chrun, I.R.: Artificial neural networks applied in the solution of the inverse kinematics problem of a 3D manipulator arm. In: International joint conference on neural networks (IJCNN) (pp. 1–6). IEEE, 2018
41. Dereli, S.: A new modified grey wolf optimization algorithm proposal for a fundamental engineering problem in robotics. Neural Comput. Appl. **33**, 14119–14131 (2021)
42. Dereli, S.: A Novel approach based on average swarm intelligence to improve the whale optimization algorithm. Arab. J. Sci. Eng. **47**, 1763–1776 (2022)
43. Lee, H.; Kim, K.; Kwon, Y.; Hong, E.: Real-time particle swarm optimization on FPGA for the optimal message-chain structure. Electronics (2018). https://doi.org/10.3390/electronics7110274
44. Li, Y.; Liu, Z.;Xu, K.; Yu, H.; Ren, F.: A 7.663-TOPS 8.2-W energy-efficient FPGA accelerator for binary convolutional neural networks. In: FPGA, pp. 290–291, 2017
45. Karakuzu, C.; Karakaya, F.; Çavuşlu, M.A.: FPGA implementation of neuro-fuzzy system with improved PSO learning. Neural Netw. **79**, 128–140 (2016)
46. Li, R.; Huang, X.; Tian, S.; Hu, R.; He, D.; Gu, Q.: FPGA-based design and implementation of real-time robot motion planning. In International Conference on Information Science and Technology (ICIST), IEEE